12장. 함수형 프로그래밍과 람다

12.1. 함수형 프로그래밍이란?

12.1.1. 함수형 프로그래밍 정의

프로그래밍 패러다임으로서의 함수형 프로그래밍

- 절차지향 프로그래밍 : 알고리즘과 로직을 중심으로 문제를 해결하기 위함을 주 목적으로 하는 프로그래밍 기법. 대표적 언어가 C, Pascal 등
-
- 객체지향 프로그래밍 : 클래스 선언을 최 우선으로 하는 프로그래밍 기법. 데이터와 데이터를 처리할 메서드를 한데 묶어 객체를 만들고 객체를 조합해서 프로그래밍 작성. 객체를 만들기 위한 추상화, 캡슐화, 상속성, 다형성등의 다양한 개념 제공. 대표적 언어로 자바, C++등
-
- 함수형 프로그래밍 : 함수 선언을 최 우선으로 하는 프로그래밍 기법. 데이터의 흐름에 초점을 마추지 않고 함수의 선언과 선언된 함수의 유기적인 흐름을 주 목적으로 함. 대표적인 언어로 코틀린, 스칼라, 스위프트 등

함수형 프로그래밍의 주요 원칙

- First Class Citizen 으로서의 함수
- 순수함수 (Pure Function) 으로 정의되는 함수

12.1. 함수형 프로그래밍이란?

함수형 프로그래밍에서의 데이터

- 데이터는 변경되지 않으며 프로그램의 상태만을 표현한다. (immutable state로서의 데이터)
- 함수에서 데이터는 변경하는 것이 아니라 새로운 데이터를 만들어 리턴한다.

함수형 프로그래밍의 이점

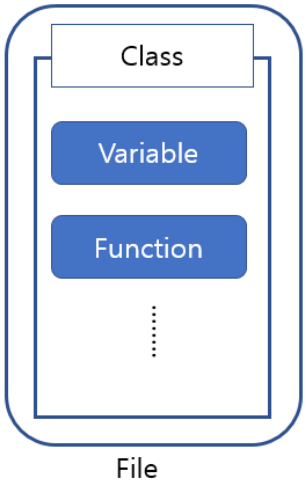
- 코드가 간결하여 개발생산성 및 유지보수성이 증대된다.
- 동시성 작업을 좀더 쉽고 안전하게 구현할수 있다.

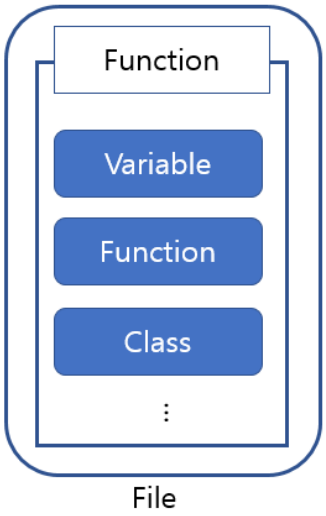
12.1. 함수형 프로그래밍이란?

12.1.2. 함수형 프로그래밍의 원칙

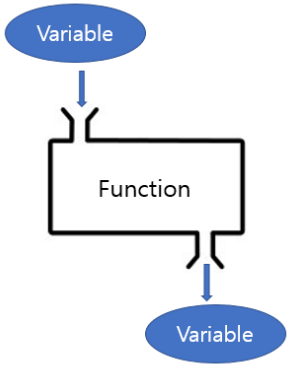
일급 객체로서의 함수

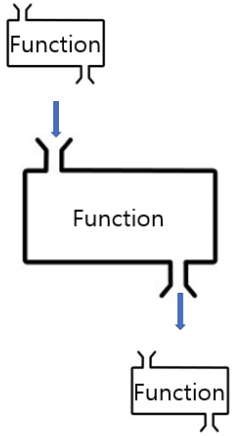
- 프로그램의 top – level에 함수를 정의
- 모든 구성요소를 함수안에 작성
- 함수를 변수처럼 이용





12.1. 함수형 프로그래밍이란?





12.1. 함수형 프로그래밍이란?

순수 함수로 정의되는 함수

- 순수 함수는 Side-Effect이 발생하지 않는 함수

- 함수 외부의 다른 값을 변경하지 않는다.
- 함수 내부에서 별도의 입력/출력이 발생하지 않는다. (파일, 데이터베이스, 네트웍등)

```
fun some(a: Int): Int {
    return (Math.random()*a).toInt()
}
```

```
fun some1(a: String): Boolean {
    try {
        val file=File("a.txt")
        val out=FileWriter(file);
        out.write(a)
        out.flush()
        return true
    }catch (e: Exception){
        return false
    }
}
```

12.1. 함수형 프로그래밍이란?

```
fun some2(a: Int, b: Int): Int{
    return a+b
}
```

```
fun some3(): Int{
    return 10
}
```

```
fun some4(a: Int, b: Int) {
    val result=a + b
}
```

12.1. 함수형 프로그래밍이란?

12.1.3. 코틀린에서 일급 객체로서의 함수

다양한 구성요소를 포함하는 함수

```
fun superFun(){
    val superData="hello"
    fun subFun1(){
        println("subFun1() .. superData : ${superData}")
    }
    fun subFun2(a: Int, b: Int): Int{
        subFun1()
        return a+b
    }

    class SubClass {
        fun classFun(){
            println("classFun() .. superData : ${superData}")
        }
    }
    subFun1()
    SubClass().classFun()
}
```

12.1. 함수형 프로그래밍이란?

변수처럼 이용되는 함수

```
val dataVal=10

val funVal=fun someFun() {//error
    //.....
}
```

```
val funVal1= { x1: Int ->
    println("hello world")
    x1 * 10
}
funVal1(10)
```

```
fun someFun() {
    println("i am some Function")
}

val funVal2=::someFun

funVal2()
```

12.2. 람다 표현식

12.2.1. 람다 표현식이란?

- 익명함수(Anonymous Functions)를 지칭하는 용어

- fun 함수이름(매개변수) { 함수내용 }
- { 매개변수 -> 함수내용 }

- 람다함수는 항상 { }에 의해 감싸 표현되어야 한다.
- { }안에 -> 표시가 있으며 -> 왼쪽은 매개변수, 오른쪽은 함수내용이 위치한다.
- 매개변수 타입은 선언되어 있어야 하며 추론이 가능한 경우에는 생략이 가능하다.
- 함수의 리턴 값은 함수내용의 마지막 표현식이다.

12.2. 람다 표현식

```
fun sum(x1: Int, x2: Int): Int {
    return x1+x2
}
```

```
val sum1={ x1: Int, x2: Int -> x1+x2 }

fun main(args: Array<String>) {
    val result1=sum1(10, 20)
}
```

- 람다함수를 정의하자 마자 함수 호출

```
{ println("hello")}()
run { println("world") }
```

12.2. 람다 표현식

- 매개변수가 없는 람다함수 정의

```
val sum2 = { -> 10 + 20 }
```

```
val sum2 = { 10 + 20 }
```

- 함수 내용이 여러 문장으로 된 경우의 리턴값

```
val sum3 = { x1:Int, x2: Int ->
    println("call sum3()....")
    x1 + x2
}
```

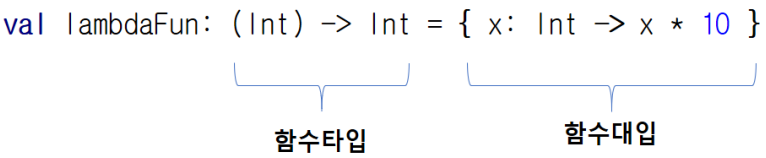
12.2. 람다 표현식

12.2.2. 함수 타입

```
val name: String = "kkang"
val no: Int = 10
```

```
fun myFun(x1: Int, x2: Int): Boolean {
    return x1 > x2
}
```

```
val lambdaFun: (Int) -> Int = { x: Int -> x * 10 }
```



12.2. 람다 표현식

Typealias를 이용한 타입 정의

```
typealias MyType = (Int) -> Boolean
```

```
val myFun: MyType = { it > 10 }
```

매개변수 타입 생략

```
val lambdaFun1 =  { x -> x * 10 }//error

val lambdaFun2: (Int) -> Int = { x -> x+ 10 }
```

12.2. 람다 표현식

12.2.3. it을 이용한 매개변수 이용

- 매개변수 하나인 경우에는 함수 내용에서 it으로 매개변수를 지칭

```
val lambdaFun3: (Int) -> Int = { x -> x+10 }

val lambdaFun4: (Int) -> Int = { it + 10  }
```

```
val itTest2 = { it * 10 }//it 에러..
```

12.2. 람다 표현식

12.2.4. 맴버 참조 이용

- Member Reference는 콜론 두개(::) 을 이용하여 타입을 명시하는 기법

```
fun main(args: Array<String>) {

    class User(val name: String, val age: Int)

    val lambdas1: (User) -> Int = { user: User -> user.age}
    println("lambdas1 return : ${lambdas1(User("kkang", 33))}")
}
```

```
val lambdas2: (User) -> Int = { it.age }
println("lambdas2 return : ${lambdas2(User("kkang", 33))}")
```

```
val lambdas3: (User) -> Int = User::age
println("lambdas3 return : ${lambdas3(User("kkang", 33))}")
```